Ecma/TC39/2015/025

ISO/IEC JTC 1/SC 22

First edition
2015-mm-dd

ISO/IEC XXXXX:2015 (E)

ISO/IEC/JTC 1/SC 22

Secretariat: ANSI

Information technology — Programming languages, their environments and system software interfaces — The JSON Data Interchange Format

Document type: International standard
Document subtype: if applicable
Document stage: (20) Preparation
Document language: E

Copyright notice

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC xxxxx was prepared by Ecma International (as ECMA-404) and was adopted, under a special “fast-track procedure”, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Introduction

JSON is a text format that facilitates structured data interchange between all programming languages. JSON is syntax of braces, brackets, colons, and commas that is useful in many contexts, profiles, and applications. JSON was inspired by the object literals of JavaScript aka ECMAScript as defined in the ECMAScript Language Specification, third Edition [1]. It does not attempt to impose ECMAScript's internal data representations on other programming languages. Instead, it shares a small subset of ECMAScript's textual representations with all other programming languages.

JSON is agnostic about numbers. In any programming language, there can be a variety of number types of various capacities and complements, fixed or floating, binary or decimal. That can make interchange between different programming languages difficult. JSON instead offers only the representation of numbers that humans use: a sequence of digits. All programming languages know how to make sense of digit sequences even if they disagree on internal representations. That is enough to allow interchange.

JSON text is a sequence of Unicode code points. JSON also depends on Unicode in the hex numbers used in the `\u` escapement notation.

Programming languages vary widely on whether they support objects, and if so, what characteristics and constraints the objects offer. The models of object systems can be wildly divergent and are continuing to evolve. JSON instead provides a simple notation for expressing collections of name/value pairs. Most programming languages will have some feature for representing such collections, which can go by names like `record`, `struct`, `dict`, `map`, `hash`, or `object`.

JSON also provides support for ordered lists of values. All programming languages will have some feature for representing such lists, which can go by names like `array`, `vector`, or `list`. Because objects and arrays can nest, trees and other complex data structures can be represented. By accepting JSON's simple convention, complex data structures can be easily interchanged between incompatible programming languages.

JSON does not support cyclic graphs, at least not directly. JSON is not indicated for applications requiring binary data.

It is expected that other standards will refer to this one, strictly adhering to the JSON text format, while imposing restrictions on various encoding details. Such standards may require specific behaviours. JSON itself specifies no behaviour.

Because it is so simple, it is not expected that the JSON grammar will ever change. This gives JSON, as a foundational notation, tremendous stability. JSON was first presented to the world at the `JSON.org` website in 2001. JSON stands for JavaScript Object Notation.

Information technology — Programming languages, their environments and system software interfaces — The JSON Data Interchange Format

1 Scope

JSON is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript programming language, but is programming language independent. JSON defines a small set of structuring rules for the portable representation of structured data.

2 Conformance

Conforming JSON text is a sequence of Unicode code points that strictly conforms to the JSON grammar.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 10646:2012*, Information Technology – Universal Coded Character Set (UCS)*

The Unicode Consortium. The Unicode Standard, Version 6.2.0, (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-07-8)

http://www.unicode.org/versions/Unicode6.2.0/.

4 JSON Text

A JSON text is a sequence of tokens formed from Unicode code points that conforms to the JSON value grammar. The set of tokens includes six structural tokens, strings, numbers, and three literal name tokens.

The six structural tokens:

`[` U+005B left square bracket

`{` U+007B left curly bracket

`]` U+005D right square bracket

`}` U+007D right curly bracket

`:` U+003A colon

`,` U+002C comma

These are three literal name tokens:

`true` U+0074 U+0072 U+0075 U+0065

`false` U+0066 U+0061 U+006c U+0073 U+0065

`null` U+006E U+0075 U+006C U+006C

Insignificant whitespace is allowed before or after any token. The whitespace characters are: character tabulation (U+0009), line feed (U+000A), carriage return (U+000D), and space (U+0020). Whitespace is not allowed within any token, except that space is allowed in strings.

5 JSON Values

A JSON value can be an *object*, *array*, *number*, *string*, `true`, `false`, or `null`.

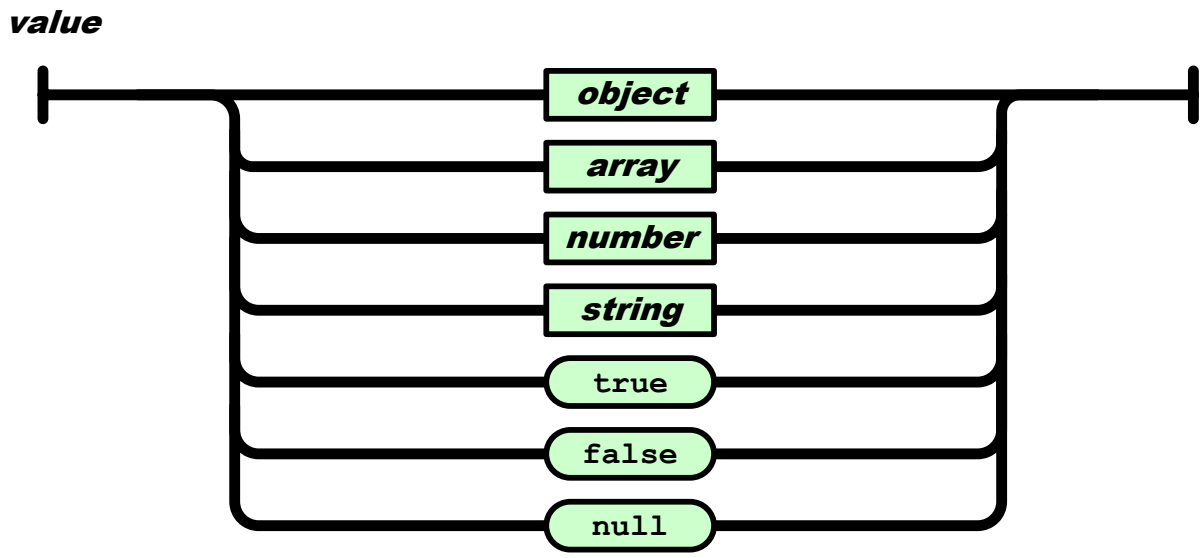


Figure 1 — value

6 Objects

An object structure is represented as a pair of curly bracket tokens surrounding zero or more name/value pairs. A name is a *string*. A single colon token follows each name, separating the name from the *value*. A single comma token separates a *value* from a following name.

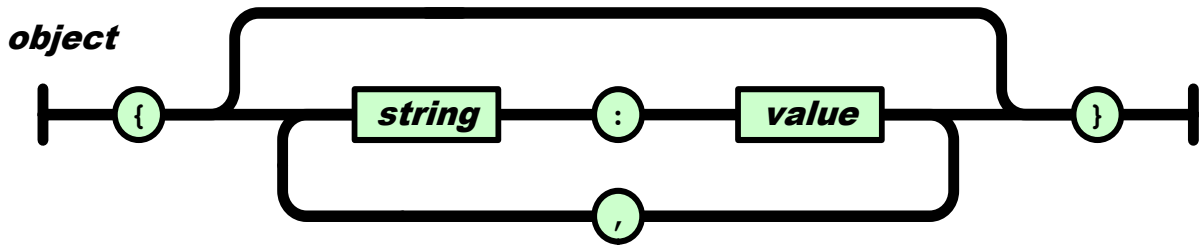


Figure 2 — object

7 Arrays

An array structure is a pair of square bracket tokens surrounding zero or more *values*. The *values* are separated by commas. The order of the *values* is significant.

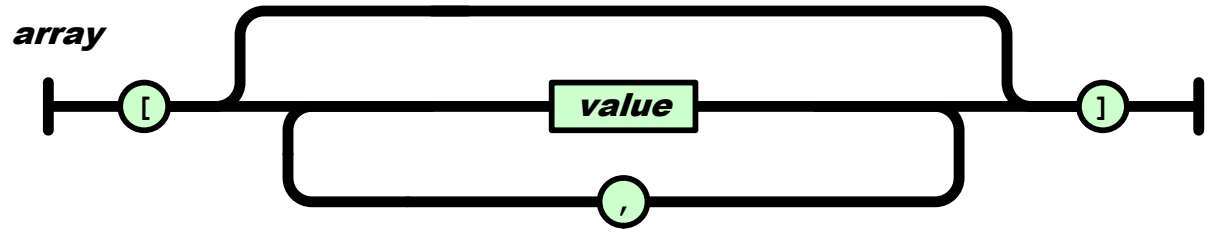


Figure 3 — array

8 Numbers

A number is represented in base 10 with no superfluous leading zero. It may have a preceding minus sign (U+002D). It may have a `.` (U+002E) prefixed fractional part. It may have an exponent of ten, prefixed by `e` (U+0065) or `E` (U+0045) and optionally `+` (U+002B) or `−` (U+002D). The digits are the code points U+0030 through U+0039.

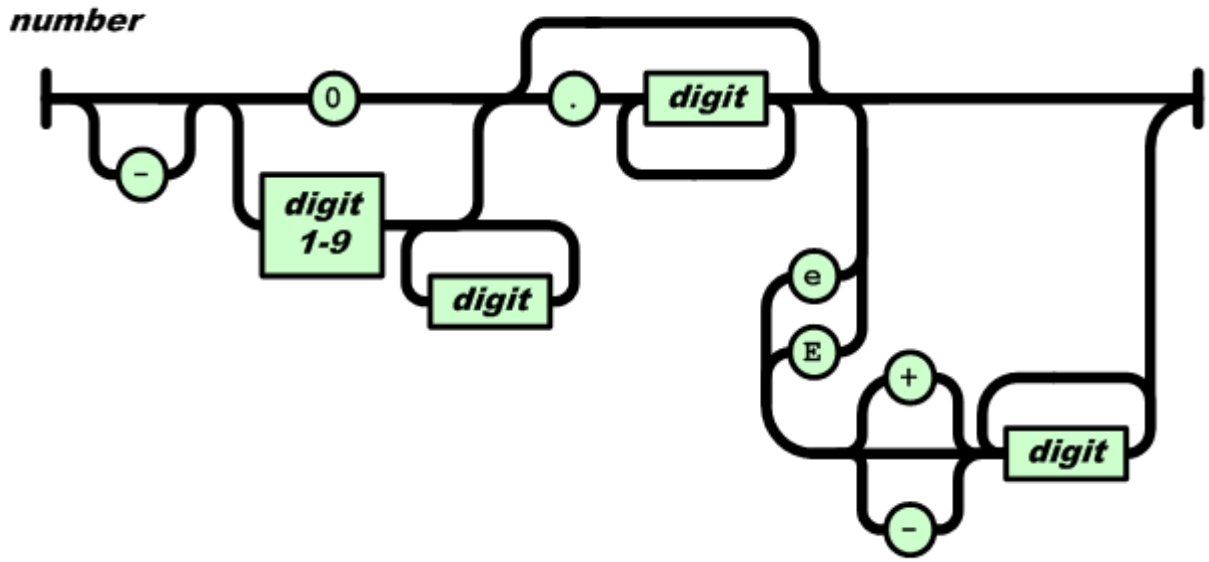


Figure 4 — number

Numeric values that cannot be represented as sequences of digits (such as `Infinity` and `NaN`) are not permitted.

9 String

A string is a sequence of Unicode code points wrapped with quotation marks (U+0022). All characters may be placed within the quotation marks except for the characters that must be escaped: quotation mark (U+0022), reverse solidus (U+005C), and the control characters U+0000 to U+001F. There are two-character escape sequence representations of some characters.

`\"` represents the quotation mark character (U+0022).

`\\` represents the reverse solidus character (U+005C).

`\/` represents the solidus character (U+002F).

`\b` represents the backspace character (U+0008).

`\f` represents the form feed character (U+000C).

`\n` represents the line feed character (U+000A).

`\r` represents the carriage return character (U+000D).

`\t` represents the character tabulation character (U+0009).

So, for example, a string containing only a single reverse solidus character may be represented as `"\\"`.

Any code point may be represented as a hexadecimal number. The meaning of such a number is determined by ISO/IEC 10646. If the code point is in the Basic Multilingual Plane (U+0000 through U+FFFF), then it may be represented as a six-character sequence: a reverse solidus, followed by the lowercase letter `u`, followed by four hexadecimal digits that encode the code point. Hexadecimal digits can be digits (U+0030 through U+0039) or the hexadecimal letters `A` through `F` in uppercase (U+0041 through U+0046) or lowercase (U+0061 through U+0066). So, for example, a string containing only a single reverse solidus character may be represented as `"\u005C"`.

The following four cases all produce the same result:

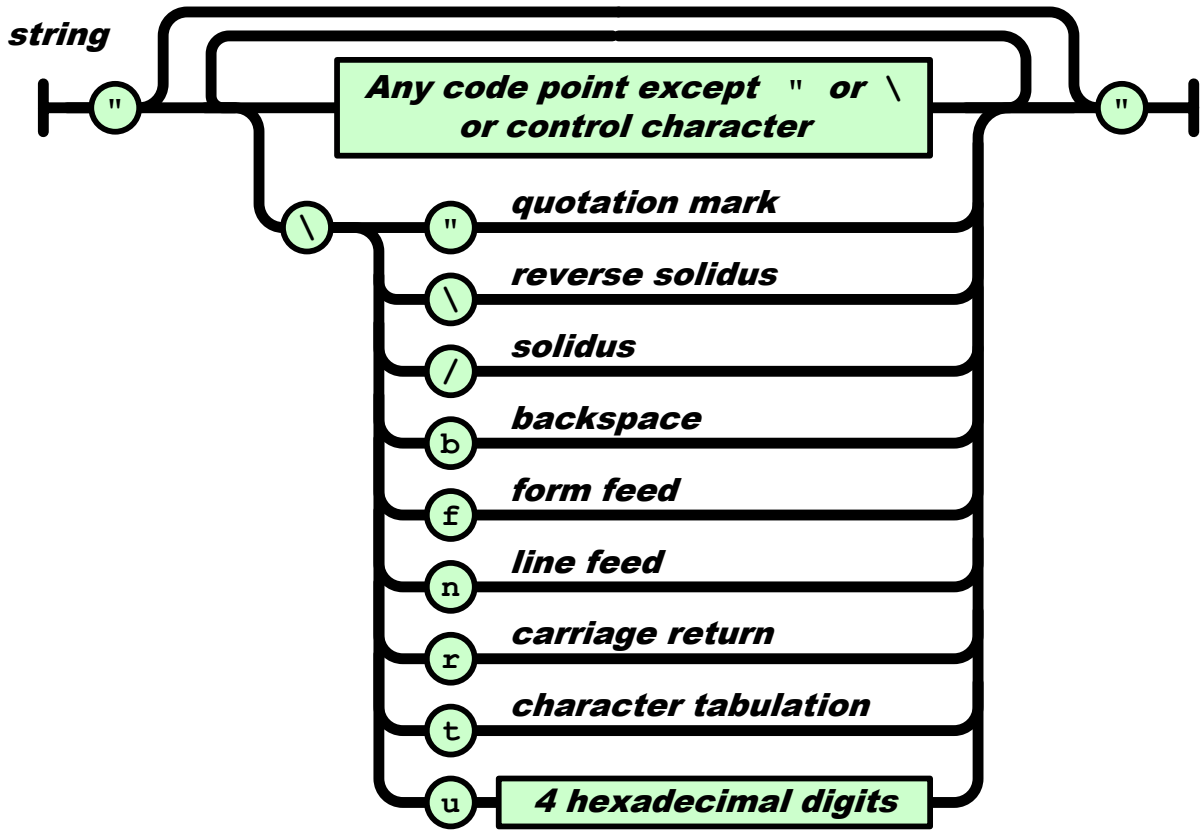
```
"\u002F"

"\u002f"

"\/"

"/"
```

To escape a code point that is not in the Basic Multilingual Plane, the character is represented as a twelve-character sequence, encoding the UTF-16 surrogate pair. So for example, a string containing only the G clef character (U+1D11E) may be represented as `"\uD834\uDD1E"`.



Bibliography

[1] ISO/IEC 16262, *Information technology — Programming languages, their environments and system software interfaces — ECMAScript® Language Specification, 3rd edition (May 2011)*